Probability Analysis for Monopoly

L. Rotgers

May 13, 2018

```
In [1]: %pylab inline
```

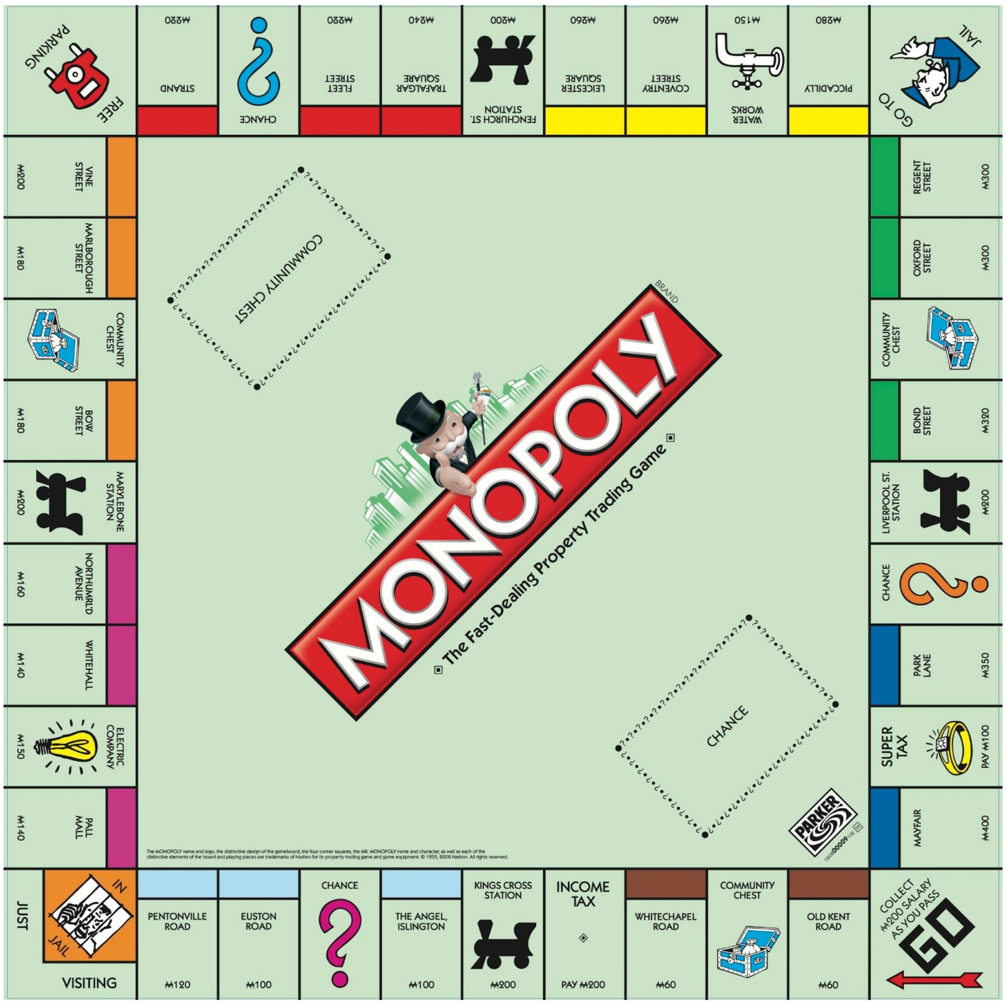
```
Populating the interactive namespace from numpy and matplotlib
```

Contents

1 Monopoly

Analysis for the probabilities with Monopoly to answer the question: which are the best houses to buy?

To answer this question we will create a simulated version of Monopoly and determine the probabilities to land on each square. The square with the highest probability will be the best square to have.



monopoly

1.1 Squares

Each edge has 9 positions and there are 4 edges. There are 4 corners. Which gives a total of 40 positions where a player can land. The labels are numbered starting at GO.

1.1.1 Labels

```
In [2]: squares_labels = ['start', 'b1', 'cc1', 'b2', 'it', 't1',
                          'lb1', 'c1', 'lb2', 'lb3', 'jail', 'p1',
                          'ec', 'p2', 'p3', 'ts2', 'o1', 'cc2',
                          'o2', 'o3', 'p', 'r1', 'c2', 'r2', 'r3',
                          'ts3', 'y1', 'y2', 'ww', 'y3', 'gtj',
                          'g1', 'g2', 'cc3', 'g3', 'ts4', 'c3',
                          'db1', 'st', 'db2']

        squares_total = len(squares_labels)
        print('There are {} squares.'.format(squares_total))
```

```
There are 40 squares.
```

1.1.2 Descriptions

We also want to know the proper names, so we don't have to look up the labels.

```
In [3]: squares_description = ['Start', 'Brown 1', 'Community Chest 1', 'Brown 2',
                               'Income Tax', 'Train Station 1', 'Light Blue 1',
                               'Chance 1', 'Light Blue 2', 'Light Blue 3', 'Jail',
                               'Purple 1', 'Electric Company', 'Purple 2',
                               'Purple 3', 'Train Station 2', 'Orange 1',
                               'Community Chest 2', 'Orange 2', 'Orange 3',
                               'Free Parking', 'Red 1', 'Chance 2', 'Red 2',
                               'Red 3', 'Train Station 3', 'Yellow 1', 'Yellow 2',
                               'Water Works', 'Yellow 3', 'Go to Jail', 'Green 1',
                               'Green 2', 'Community Chest 3', 'Green 3',
                               'Train Station 4', 'Chance 3', 'Dark Blue 1',
                               'Super Tax', 'Dark Blue 2']
```

```
In [4]: print('There are {} descriptions.'.format(len(squares_description)))
```

```
There are 40 descriptions.
```

1.1.3 Purchasable

We want to know if they are purchasable so we can sort on that later.

```
In [5]: squares_purchasable = [False, True, False, True, False,
                               True, True, False, True, True,
                               False, True, True, True, True,
                               True, True, False, True, True,
                               False, True, False, True, True,
                               True, True, True, True, True,
                               False, True, True, False, True,
                               True, False, True, False, True]
```

1.1.4 Grouping

We want to know in what group they are so we can aggregate our data later.

```
In [6]: squares_aggregate = ['Start', 'Brown', 'Community Chest', 'Brown',
                             'Income Tax', 'Train Station', 'Light Blue',
                             'Chance', 'Light Blue', 'Light Blue', 'Jail',
                             'Purple', 'Electric Company', 'Purple', 'Purple',
                             'Train Station', 'Orange', 'Community Chest',
                             'Orange', 'Orange', 'Free Parking', 'Red',
                             'Chance', 'Red', 'Red', 'Train Station', 'Yellow',
                             'Yellow', 'Water Works', 'Yellow', 'Go to Jail',
                             'Green', 'Green', 'Community Chest', 'Green',
                             'Train Station', 'Chance', 'Dark Blue',
                             'Super Tax', 'Dark Blue']
```

1.2 Cards

There are two decks of cards.

- Community Cards
- Chance Cards

Each deck contains 16 cards.

1.2.1 Community cards

Monopoly has 16 community cards.

cc

Because we are only determining the probabilities, we are only interested in the following cards:

- advance to go
- go to jail
- get out of jail, free
- go back 2 spaces

1.2.2 Community deck implementation

We implement the community deck in a class. The class keeps track of a list with 16 cards. An index points to the next card. When we are out of cards, we reset the index and reshuffle the cards.

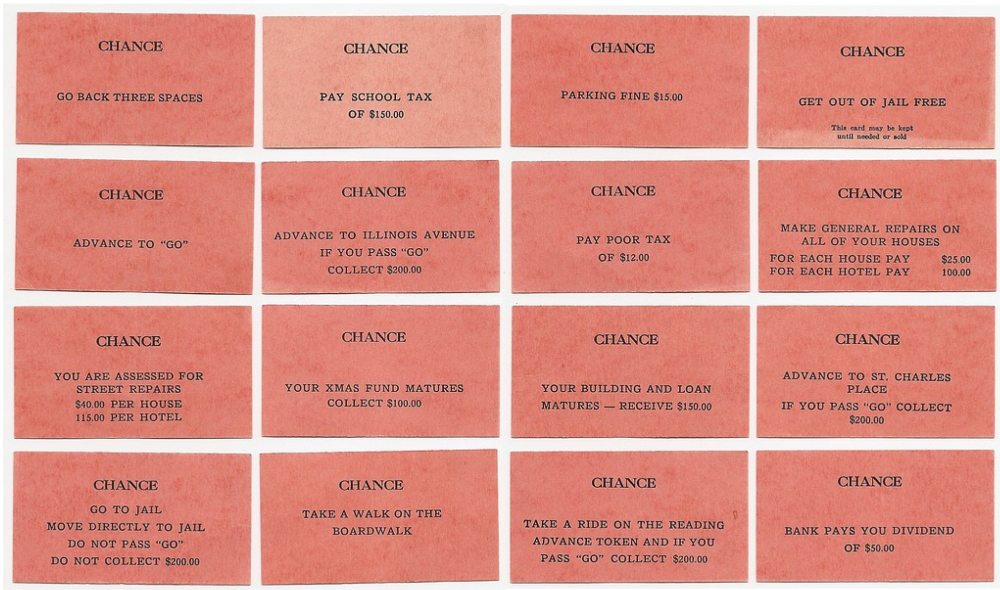
```
In [28]: from random import shuffle

         class CommunityDeck():
             def __init__(self):
                 self.deck = [0] * 16
                 self.deck[0] = 'gtg' # go to go
                 self.deck[1] = 'gtj' # go to jail
                 self.deck[2] = 'goj' # get out of jail
                 self.deck[3] = 'gb2' # go back 2 steps
                 self.index = 16

             def draw_card(self):
                 if self.index >= len(self.deck):
                     self.index = 0
                     shuffle(self.deck)
                 card = self.deck[self.index]
                 self.index += 1
                 return card
```

Now we test it:

```
In [34]: deck = CommunityDeck()
         deck.deck

Out[34]: ['gtg', 'gtj', 'goj', 'gb2', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

In [35]: deck.draw_card()

Out[35]: 0

In [36]: deck.deck

Out[36]: [0, 0, 0, 0, 'gb2', 0, 0, 'gtg', 0, 'gtj', 0, 'goj', 0, 0, 0, 0]

In [37]: deck.index

Out[37]: 1
```

1.2.3 Chance cards

Monopoly has 16 chance cards.



chance

Because we are only determining the probabilities, we are only interested in the following cards:

- go back three spaces
- get out of jail free
- advance to go
- advance to illinois avenue (R3)
- go to jail

1.2.4 Chance deck implementation

We implement the chance deck in a class. The class keeps track of a list with 16 cards. An index points to the next card. When we are out of cards, we reset the index and reshuffle the cards.

```
In [8]: from random import shuffle

        class ChanceDeck():
            def __init__(self):
                self.deck = [0] * 16
                self.deck[0] = 'gtg' # go to go
                self.deck[1] = 'gtj' # go to jail
                self.deck[2] = 'goj' # get out of jail
                self.deck[3] = 'gb3' # go back 3
                self.deck[4] = 'r3'  # go to red 3 (r3)
                self.index = 16

            def draw_card(self):
                if self.index >= len(self.deck):
                    self.index = 0
                    shuffle(self.deck)
                card = self.deck[self.index]
                self.index += 1
                return card
```

Now we test it:

```
In [30]: deck = ChanceDeck()
         deck.deck

Out[30]: ['gtg', 'gtj', 'goj', 'gb3', 'r3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

In [31]: deck.draw_card()

Out[31]: 0

In [32]: deck.deck

Out[32]: [0, 'gtj', 0, 'gb3', 0, 0, 0, 0, 0, 0, 'goj', 0, 0, 'gtg', 0, 'r3']

In [33]: deck.index

Out[33]: 1
```

1.3 Dice

We will be implementing the dice as a class. This allows us to encapsulate how the result is determined. It makes it easier to implements other scenarios such as throwing with multiple dices.

```
In [9]: from random import randint

        class Dice():
            def __init__(self, dices = 1, sides = 6):
                self.dices = dices
                self.sides = 6

            def throw(self):
                total = 0
                for i in range(self.dices):
                    total += randint(1, self.sides)
                return total
```

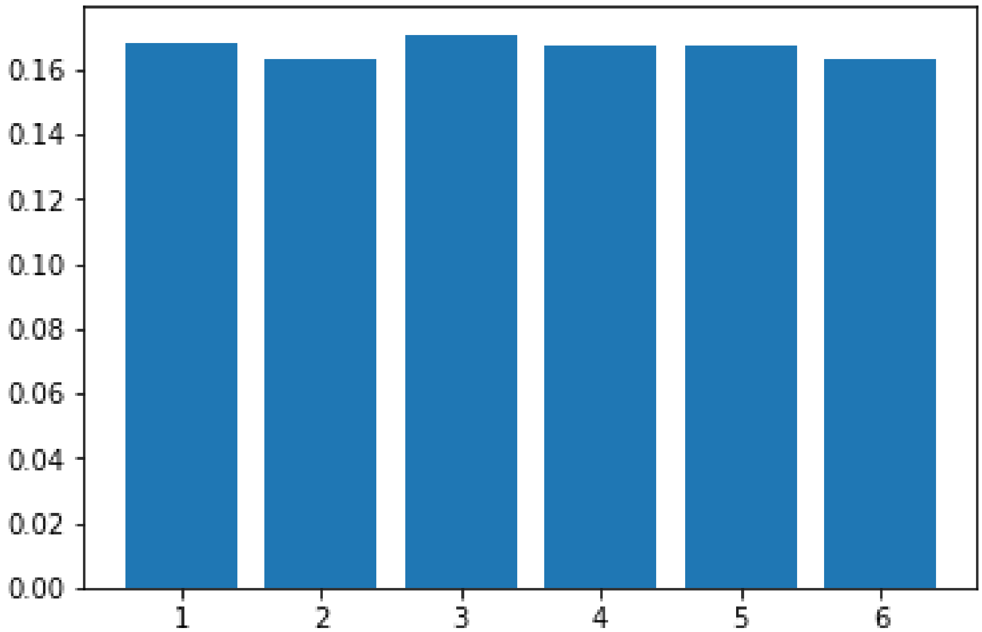
Rolling one time:

```
In [39]: dice = Dice()
         dice.throw()

Out[39]: 5
```

1.3.1 Simple: one dice with six sides

A simple setup would be one dice with six sides. This will give uniformly distributed probabilities.

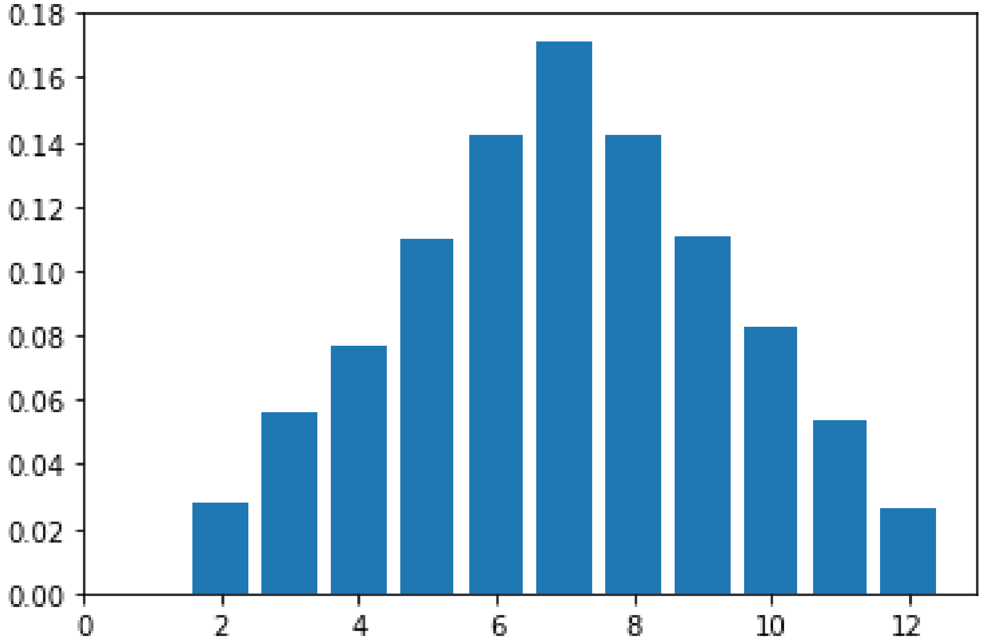
```
In [10]: dice = Dice()
         sides = [0] * dice.dices * dice.sides
         N = 10000
         for i in range(N): sides[dice.throw()-1] += 1
         sides = np.array(sides) / N
         bar(range(1,len(sides)+1), sides);
```



1.3.2 Advanced: two dice with four sides

A more advanced set up would be to play with two dices and four sides. This will give normally distributed probabilities.

```
In [11]: dice = Dice(2,4)
         sides = [0] * dice.dices * dice.sides
         N = 10000
         for i in range(N): sides[dice.throw()-1] += 1
         sides = np.array(sides) / N
         bar(range(1,len(sides)+1), sides);
```



1.4 Monopoly simulation

1.4.1 Algorithm

Here we are going to simulate a game for N amount of rounds. The game algorithm is simple:

1. Roll the dice
2. Move to the new position
3. Increment the square counter for that position
4. Check and handle go to jail
5. Check and handle community chest
6. Check and handle chance

1.4.2 Modulo arithmetic for position tracking

We can easily keep track of our position with modulo arithemetic. Let C be our position (or `index`), d the result from throwing the dice, and n the current round. To determine our new position we calculate:

$$C_{n+1} \equiv C_n + d \pmod{40}$$

The modulo is 40 because that are the total amount of squares.

1.4.3 Implementation

Below is the implementation for the Monopoly simulation.

```
In [12]: dice = Dice()
         community_deck = CommunityDeck()
         chance_deck = ChanceDeck()

         index = 0 # position
         total_squares = len(squares_labels)
         squares = [0] * total_squares

         rounds = 1000000

         for i in range(rounds):

             # Throw the dice and move our position on the board.
             steps = dice.throw()
             index = (index + steps) % total_squares
             squares[index] += 1

             # We landed on go to jail.
             if squares_labels[index] is 'gtj':
                 index = squares_labels.index('jail')

             # We landed on the community card.
             if squares_labels[index] in ['cc1', 'cc2', 'cc3']:
                 card = community_deck.draw_card()
                 if card is 'gtg': index = squares_labels.index('start')
                 if card is 'gtj': index = squares_labels.index('jail')
                 if card is 'gb2':
                     if index >= 2: index -= 2
                     if index < 2: index = total_squares-abs(index-2)-1

             # We landed on the chance card.
             if squares_labels[index] in ['c1', 'c2', 'c3']:
                 card = chance_deck.draw_card()
                 if card is 'gtg': index = squares_labels.index('start')
                 if card is 'gtj': index = squares_labels.index('jail')
                 if card is 'r3': index = squares_labels.index('r3')
                 if card is 'gb3':
                     if index >= 3: index -= 3
                     if index < 3: index = total_squares-abs(index-3)-1
```

It takes around 2.7 seconds to run a game when $N = 1,000,000$.

1.5 Detailed probability analysis

Now we can proceed to analyze our results.

1.5.1 Determining probabilities

With the number of times that each square is visited we can calculate the probabilities. The probability that a square is visited is:

$$P(\bar{x} = x) = \frac{\text{Times visited}}{\text{\# of rounds}}$$

We also want to create a `DataFrame` in Python to easily keep track of everything.

```
In [13]: import pandas as pd
         df = pd.DataFrame(index=range(total_squares))
         df['Square'] = squares_labels
         df['Description'] = squares_description
         df['Purchasable'] = squares_purchasable
         df['Visited'] = squares
         df['Probability'] = df['Visited'] / rounds
         df['Aggregate'] = squares_aggregate
```

We can calculate a quick summary about the data:

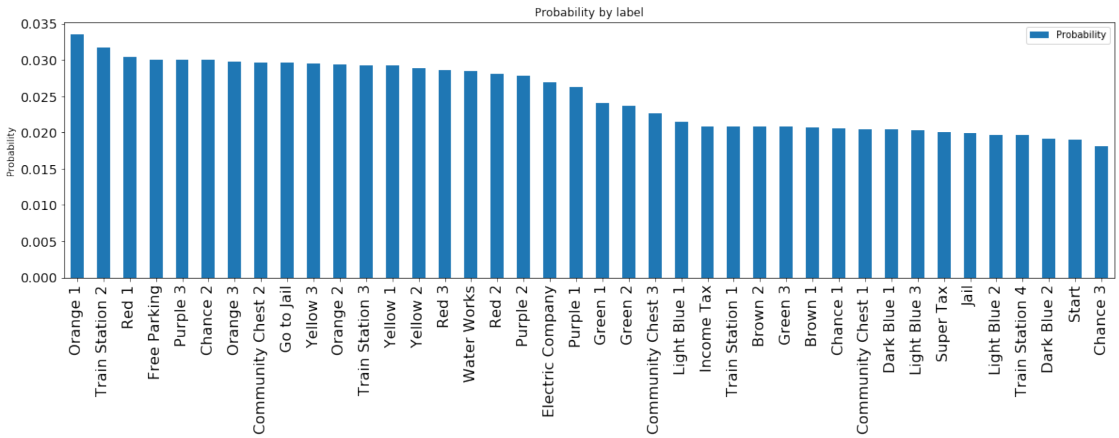
```
In [14]: print('Total rounds: {}'.format(rounds))
         print('Visited avg:  {}'.format(df['Visited'].mean()))
         print('Visited min:  {}'.format(df['Visited'].min()))
         print('Visited max:  {}'.format(df['Visited'].max()))
         print('Visited std:  {:.2f}'.format(df['Visited'].std()))
```

```
Total rounds: 1000000
Visited avg:  25000.0
Visited min:  18140
Visited max:  33538
Visited std:  4644.91
```

1.5.2 Plot of probabilities by square

If we sort these values in descending order on the probability, we can easily see which squares have the highest probability to be visited.

```
In [45]: plt.rc('xtick', labelsize=16)
         plt.rc('ytick', labelsize=14)
         df[['Description', 'Probability']].sort_values(by='Probability', ascending=False)\
             .plot(kind='bar', figsize=(20,5))
         plt.xticks(range(total_squares), df[['Description', 'Probability']]
             .sort_values(by='Probability', ascending=False)['Description'])
         plt.ylabel('Probability')
         plt.title('Probability by label');
```



Here we can conclude that `Orange 1` is the most visited square. Also notice that `Orange 2` and `Orange 3` are pretty high. It seems that `Orange` is the best street to have.

1.5.3 Table of probabilities by square

Below is the full table with all the squares and their corresponding values.

```
In [25]: df.loc[:, 'Square':'Probability'].sort_values(by='Probability', ascending=False)
```

Out[25]:

	Square	Description	Purchasable	Visited	Probability
16	o1	Orange 1	True	33538	0.033538
15	ts2	Train Station 2	True	31676	0.031676
21	r1	Red 1	True	30453	0.030453
20	p	Free Parking	False	30063	0.030063
14	p3	Purple 3	True	30037	0.030037
22	c2	Chance 2	False	29980	0.029980
19	o3	Orange 3	True	29722	0.029722
17	cc2	Community Chest 2	False	29637	0.029637
30	gtj	Go to Jail	False	29563	0.029563
29	y3	Yellow 3	True	29470	0.029470
18	o2	Orange 2	True	29360	0.029360
25	ts3	Train Station 3	True	29204	0.029204
26	y1	Yellow 1	True	29178	0.029178
27	y2	Yellow 2	True	28880	0.028880
24	r3	Red 3	True	28651	0.028651
28	ww	Water Works	True	28474	0.028474
23	r2	Red 2	True	28010	0.028010
13	p2	Purple 2	True	27773	0.027773
12	ec	Electric Company	True	26916	0.026916
11	p1	Purple 1	True	26309	0.026309
31	g1	Green 1	True	24060	0.024060
32	g2	Green 2	True	23654	0.023654
33	cc3	Community Chest 3	False	22656	0.022656

6	lb1	Light Blue 1	True	21486	0.021486
4	it	Income Tax	False	20836	0.020836
5	t1	Train Station 1	True	20824	0.020824
3	b2	Brown 2	True	20812	0.020812
34	g3	Green 3	True	20763	0.020763
1	b1	Brown 1	True	20650	0.020650
7	c1	Chance 1	False	20565	0.020565
2	cc1	Community Chest 1	False	20450	0.020450
37	db1	Dark Blue 1	True	20430	0.020430
9	lb3	Light Blue 3	True	20238	0.020238
38	st	Super Tax	False	20028	0.020028
10	jail	Jail	False	19892	0.019892
8	lb2	Light Blue 2	True	19715	0.019715
35	ts4	Train Station 4	True	19699	0.019699
39	db2	Dark Blue 2	True	19163	0.019163
0	start	Start	False	19045	0.019045
36	c3	Chance 3	False	18140	0.018140

1.5.4 Top 10 highest probability squares

The top 10 squares that have the highest probability for a player to land on are:

```
In [46]: df.loc[df['Purchasable'] == True, 'Square':'Probability']\
             .sort_values('Probability', ascending=False).head(10)
```

Out[46]:

	Square	Description	Purchasable	Visited	Probability
16	o1	Orange 1	True	33538	0.033538
15	ts2	Train Station 2	True	31676	0.031676
21	r1	Red 1	True	30453	0.030453
14	p3	Purple 3	True	30037	0.030037
19	o3	Orange 3	True	29722	0.029722
29	y3	Yellow 3	True	29470	0.029470
18	o2	Orange 2	True	29360	0.029360
25	ts3	Train Station 3	True	29204	0.029204
26	y1	Yellow 1	True	29178	0.029178
27	y2	Yellow 2	True	28880	0.028880

The total probability for all 10 squares is:

```
In [47]: df.loc[df['Purchasable'] == True].sort_values('Probability', ascending=False)\
             .head(10)['Probability'].sum()
```

```
Out[47]: 0.30151800000000001
```

1.6 High-level probability analysis

We want to answer the following questions:

1. What are the best streets to have?
2. What is the probability to be in jail?
3. What is the probability to draw a card?

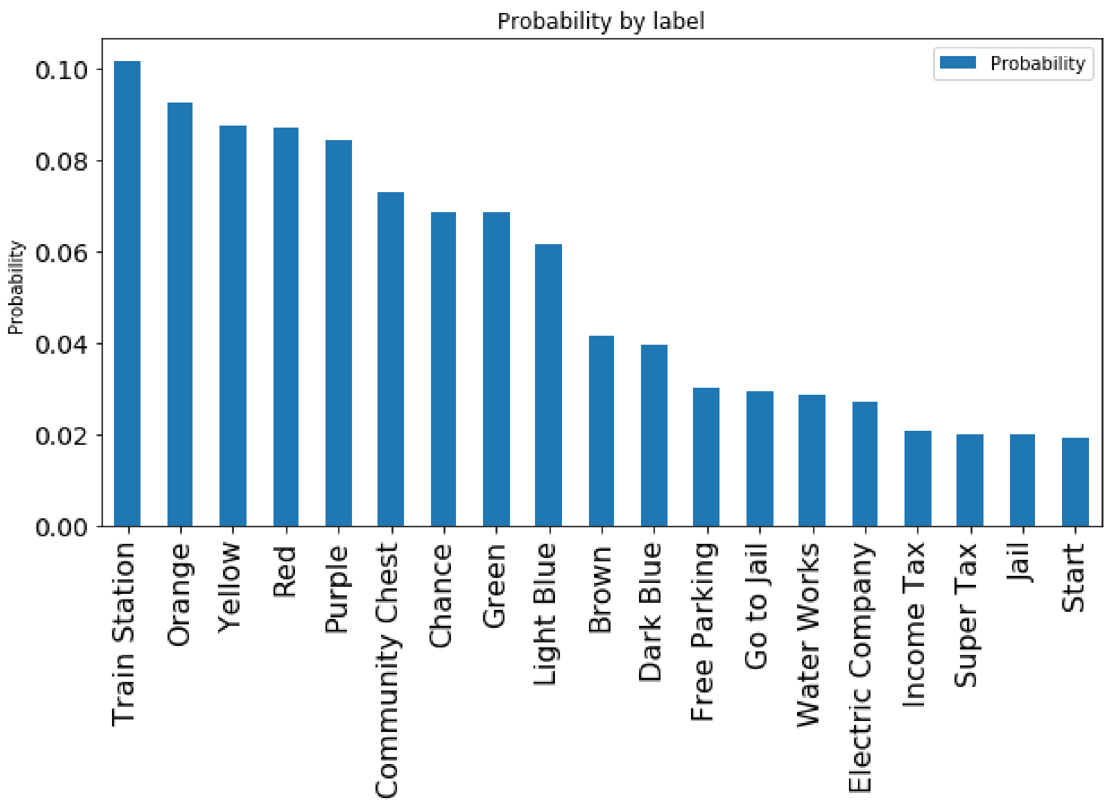
1.6.1 Aggregating (grouping by)

To find what the probabilities are per street, chance, community chest, etc., we are going to aggregate the possibilities.

```
In [19]: aggregated_df = pd.DataFrame(df.groupby(['Aggregate'])['Probability'].sum()).reset_inde
```

Now we plot the aggregated probabilities.

```
In [48]: plt.rc('xtick', labelsize=16)
         plt.rc('ytick', labelsize=14)
         aggregated_df[['Aggregate', 'Probability']].sort_values(by='Probability', ascending=Fal
             .plot(kind='bar', figsize=(10,5))
         plt.xticks(range(len(aggregated_df.index)), aggregated_df[['Aggregate', 'Probability']]
             .sort_values(by='Probability', ascending=False)['Aggregate'])
         plt.ylabel('Probability')
         plt.title('Probability by label');
```



1.6.2 Overview of all the probabilities by aggregate

A total overview of all the probabilities can be found in the table below:

```
In [21]: aggregated_df.sort_values('Probability', ascending=False)
```

```
Out[21]:          Aggregate  Probability
         16    Train Station     0.101403
         11           Orange     0.092620
         18           Yellow     0.087528
         13              Red     0.087114
         12           Purple     0.084119
         2   Community Chest     0.072743
         1            Chance     0.068685
         7             Green     0.068477
         10       Light Blue     0.061439
         0             Brown     0.041462
         3         Dark Blue     0.039593
         5      Free Parking     0.030063
         6        Go to Jail     0.029563
         17      Water Works     0.028474
         4  Electric Company     0.026916
         8        Income Tax     0.020836
         15        Super Tax     0.020028
         9              Jail     0.019892
         14            Start     0.019045
```

1.6.3 Train station probabilities

We can conclude that `Train Station` has the highest probability to land on. However, we need to take into account that there are four squares to land on.

```
In [27]: df.loc[df['Aggregate'] == 'Train Station', 'Square':'Probability'].sort_values('Probabi
```

```
Out[27]:    Square      Description  Purchasable  Visited  Probability
         15    ts2  Train Station 2         True    31676     0.031676
         25    ts3  Train Station 3         True    29204     0.029204
         5      t1  Train Station 1         True    20824     0.020824
         35    ts4  Train Station 4         True    19699     0.019699
```

1.6.4 Probability to be in jail

To find the total probability to be in jail, we need to take into account that:

- We can land on jail.
- We can land on go to jail.
- There is one community card which sends you to jail.
- There is one chance card which sends you to jail.

Each deck has 16 cards, therefore the probability to draw go to jail is $P(\bar{x} = \text{go to jail}) = \dfrac{1}{16}$.

$$P(\bar{x} = \text{in jail}) = P(\bar{x} = \text{jail}) + P(\bar{x} = \text{go to jail}) + \frac{1}{16}\left[P(\bar{x} = \text{community chest}) + P(\bar{x} = \text{chance})\right]$$

```
In [50]: P_jail           = sum(aggregated_df.loc[aggregated_df['Aggregate'] == 'Jail'
         P_go_to_jail     = sum(aggregated_df.loc[aggregated_df['Aggregate'] == 'Go to Jail'
         P_community_card = sum(aggregated_df.loc[aggregated_df['Aggregate'] == 'Community Chest
         P_chance_card    = sum(aggregated_df.loc[aggregated_df['Aggregate'] == 'Chance'
```

```
In [51]: P_jail + P_go_to_jail + 1/16 * (P_community_card + P_chance_card)
```

```
Out[51]: 0.058294249999999999
```

1.6.5 Probability to draw a card

To find the probability to draw a card, we simply calculate:

$$P(\bar{x} = \text{draw a card}) = P(\bar{x} = \text{community chest}) + P(\bar{x} = \text{chance})$$

```
In [52]: P_community_card + P_chance_card
```

```
Out[52]: 0.141428
```

Where the probabilities for the community chest square are:

```
In [53]: df.loc[df['Aggregate'] == 'Community Chest', 'Square':'Probability'].sort_values('Proba
```

Out[53]:

	Square	Description	Purchasable	Visited	Probability
17	cc2	Community Chest 2	False	29637	0.029637
33	cc3	Community Chest 3	False	22656	0.022656
2	cc1	Community Chest 1	False	20450	0.020450

And the probabilities for the chance square are:

```
In [54]: df.loc[df['Aggregate'] == 'Chance', 'Square':'Probability'].sort_values('Probability',
```

Out[54]:

	Square	Description	Purchasable	Visited	Probability
22	c2	Chance 2	False	29980	0.029980
7	c1	Chance 1	False	20565	0.020565
36	c3	Chance 3	False	18140	0.018140